ETAPE 1:
Jee WebApp, CRUD / BDD

Objectif

Arrvier à monter une application Web Jee comprenant:

- un formulaire
- agissant sur une unique table de BDD (table "Abonnés", par exemple: 4 champs Prénom, Nom, Âge, et une clé primaire) . Une seule table dans la BDD.
- Le formulaire doit permettre les opértions CRUD sur les enregistrements de la table de la BDD.

BOM:

- Tomcat,
- mariaDB,
- client HeidiSQL https://www.heidisql.com/ (client graphique SQL)
- accès "management" BDD avec le user "lauriane/lauriane"
- accès BDD par l'application Web Java Jee avec l'utilisateur:
  - username: "appli-de-lauriane"
  - mot de passe: "mdp@ppli-l@urian3"

Autres installations (hors VM) :

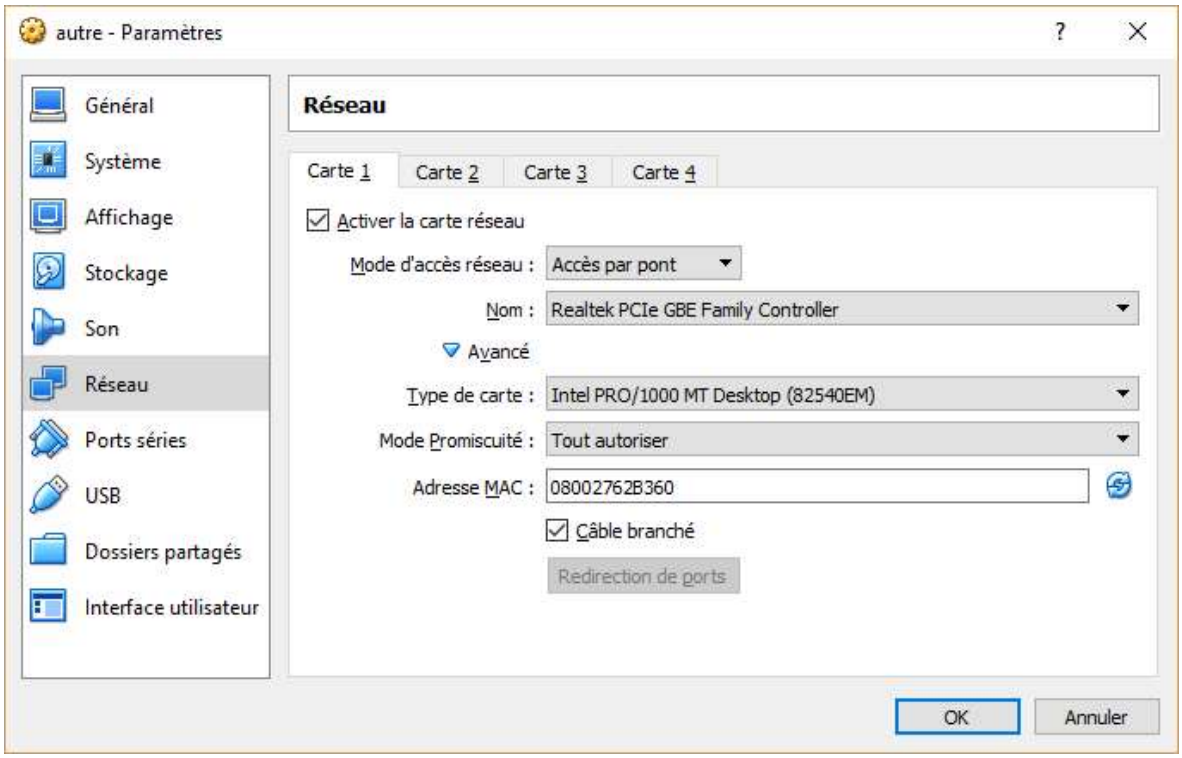
- eclipse sur mon pc/mac habituel ("hôte" de virt.).
- HeidiSQL

Mode d'emploi:

- Nous allons créer une nouvelle VM Ubuntu. Pour pouvoir réaliser l'ensemble des opérations suivantes, la VM Ubuntu que tu vas créer:
  - *doit* avoir accès à Internet.

    (ou à un réseau IP, et dans ce réseau pouvoir accéder par le protocole TCP/IP, à ce que l'on appelle des "repository" ("dépôt") valides pour le logiciel "apt-get", mais ceci est une autre histoire).

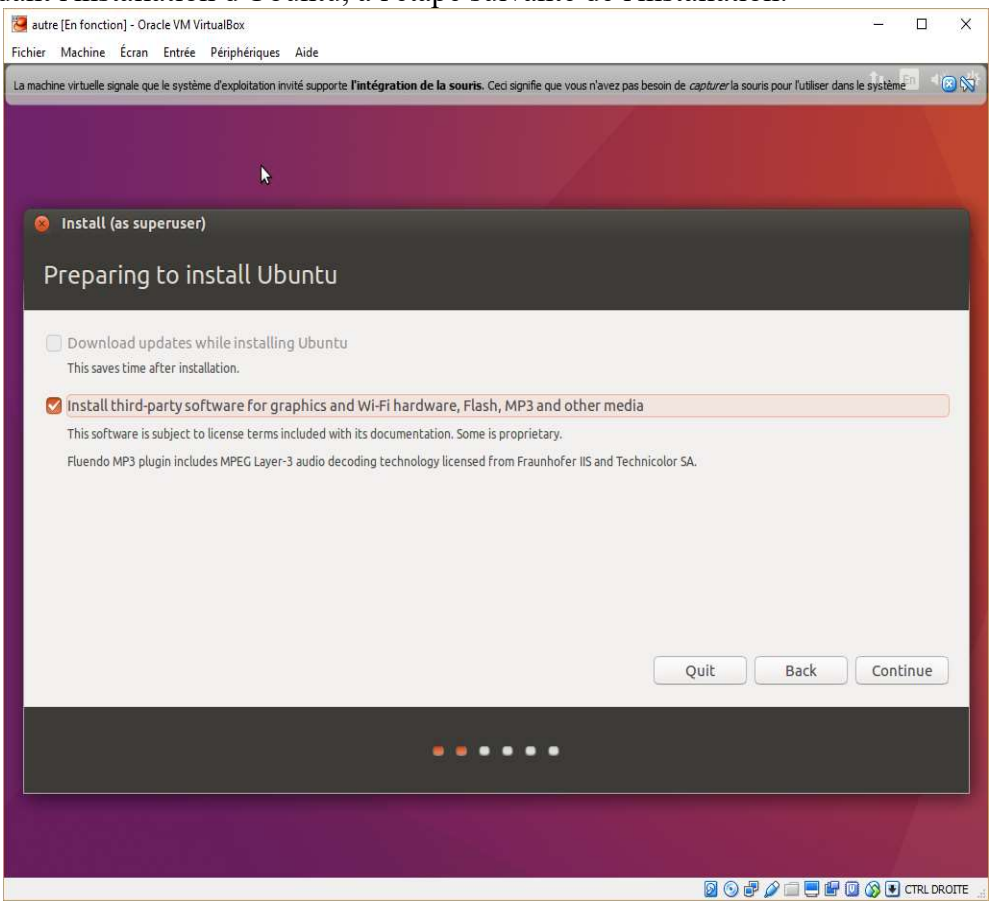
    Un test simple: si dans ta VM Ubuntu, tu as accès à internet en ouvrant firefox, et que tu connait l'adresse IP de ta "box" FAI, alors tu as ce qu'il faut.
- Crées une nouvelle VM: tu y installeras donc Ubuntu. La carte réseau virtuelle de la VM doit être configurée :
  - avec un Mode d'accès réseau de type "Bridge Network" ("Accès par pont"),
  - et un Mode Promiscuité de type "Allow all" ("Tout autoriser"):



Ci-dessus, on voit que j'ai configuré le mode d'accès réseau "Accès par pont", et le mode promiscuité "Tout autoriser" pour l'une des cartes réseau virtuelles, d'une VM Virtual Box que j'ai créée

- Tu devra installer une version bien précise d'Ubuntu, la version "16.04 LTS". Tu peux télécharger une image ISO d'installation de cette version à l'aide du lien suivant (copies-colle ce lien dans la barre d'adresse Firefox): ftp://ftp.free.fr/mirrors/ftp.ubuntu.com/releases/16.04/ubuntu-16.04.3-desktop-amd64.iso
- Si avec le PC/MAC avec lequel tu travailles, tu es connectée à internet via wifi, alors,

pendant l'installation d'Ubuntu, à l'étape suivante de l'installation:



Tu devras cocher l'option "Install third-party software for graphics and Wifi hardware [...]": lorsque cette option est cochée, un pilote ("driver") de carte WIFI sera installé, ce qui permettra à ta VM de se connecter en Wifi.
À une étape ultérieure de l'installation, tu devras créer un utilisateur linux. Tu choisiras pour cet utilisateur le nom "**lauriane-lx-usr**" et le mot de passe "**lemdp**".

- Effectue maintenant l'installation Ubuntu.
- Quand l'installation est terminée, si ton PC/MAC est connecté en WIFI à internet, il faut configurer la connexion Wifi de ta nouvelle VM. Pour t'aider dans cette configuration, reportes-toi à l' Annexe I. "Configuration WIFI de la VM".
- Lorsque l'installation Ubuntu est terminée, et que tu as accès à Internet dans ta VM, ouvres un terminal, et exécutes les commandes:

```
sudo apt-get install -y git
git clone https://github.com/Jean-Baptiste-Lasselle/lauriane
# optionnellement, pour une version spécifique
# (exemple: la release "v2.0")
# git checkout tags/v2.0
# accessoirement, il serait logique de créer d'abord une branche:
# git branch mabranchedetravail1
# pour ensuite faire le checkout dans la branche
# git checkout tags/v2.0 -b mabranchedetravail1
# afin de pouvoir faire le merge plus tard...
```

- puis les commandes:

```
sudo chmod +x lauriane/monter-cible-deploiement.sh
sudo lauriane/monter-cible-deploiement.sh
```

- À cette étape, tu as un serveur Tomcat est prêt à être utilisé. Tu peux maintenant déployer une application Web Java Jee exemple. Pour cela ouvres un seond terminal (ou une session SSH avec PUTTY), et exécute la commande:

```
sudo lauriane/deployer-appli-web.sh
```

- pour déployer "un *.war quelconque":

```
NOM_FICHER_WAR=./lauriane/nom-de-ton-fichier.war
sudo lauriane/deployer-appli-web.sh $NOM_FICHER_WAR
```

ANNEXE III. Ajout d'un routeur au lab, pour s'isoler des contraintes des résaux gérés apr d'autres.

Procédure détaillée d'installation de VyOS.

- On part de l'ISO
- 3 cartes réseaux sur la VM, toutes en accès par pont, Mode promiscuité "Tout autoriser"
- on lance la machine: on arrive à un écran où nosu demande de presser la touche entrée pour commencer l'installation
- puis on se trouve directement à devoir s'identitifer : on se connecte avec l'utilisateur/mdp "vyos/vyos", puis on exécute la commande d'installation :

```
install image
```

- l'installation est lancée, et guidée. Laisser toutes les valeurs par défaut.
- Lorsque l'installation est terminée, on fait des constats:
  - exécuter "ip addr" : permet de vérifier un point important : au contraire des distributions comme CentOS 7, ou ubuntu Server 14.04, les interfaces réseau linux configurées pendant le processus d'installation de VyOS ne sont PAS par déafut, configurées en DHCP. "ip addr" montre en effet, qu'elles n'ont pas d'adresses IP attribuées par ma "livebox", à laquelle les 3 cartes réseaux sont bien connctées puisqu'elles sont configurées pour un "Accès par pont". C'est une différence importante en "hôtes" et "routeurs", dans un réseau, et on comprendra donc pourquoi au fil des opératiosn suivantes.
  - Il est possible d'exécuter des commandes sudo, avec l'utilisateur "vyos", et ce,s ans que le mot de passe sudo soit demandé. Apr exemple "sudo cat /etc/sudoers" est possible sans donner de mot de passe.
- Après ces constats, l'installation est terminée, et on va maintenant commencer à configurer le routeur VyOS, et exécuter des commandes spécifiques à VyOS (totuefois, la commande "`install image`" utilisée précédemment, est une command spécifique VyOS). Pour commencer, on a vérifié lorsque l'on a exécuté "ip addr", que 3 interfaces réseau linux ont été créées, une pour chaque carte réseau vrituelle (de la VM VirtualBox): "eth0", "eth1", "eth2". On a de plus vérifié qu'aucune de ces interface n'a d'adresse ip. Nous allons mettre en oeuvre la topologie suivante:
  - *Sans considérer le routeur que vous êtes en train d'nstaller en suivant ce document, vous avez déjà un réseau avec lequel vous travaillez, y compris si vous n'avez pas internet: le réseau dans lequel se trouve la machien physique PC/MAC avec laquelle vous travaillez. La machien avec laquelle je travaille à la maison à l'adresse IP 192.168.1.14.*

    Si votre machine n'a pas d'adresse IP, il est possible de lui en configurer une statique.

    En tout cas, la machine physique avec laquelle vous travaillez a une adresse IP, et appartient donc à un réseau IP. Chez moi, ma machine physique appartient au réseau 192.168.1.0/24. chez moi, c'est aussi le réseau sur lequelle est accessible ma livebox, donc accès à internet, avec l'adresse IP 192.168.1.1

    Ma machine physique appartient au réseau 192.168.1.0/24, et sur ce réseau, l'adresse IP du routeur qui donen accès à internet est 192.168.1.1

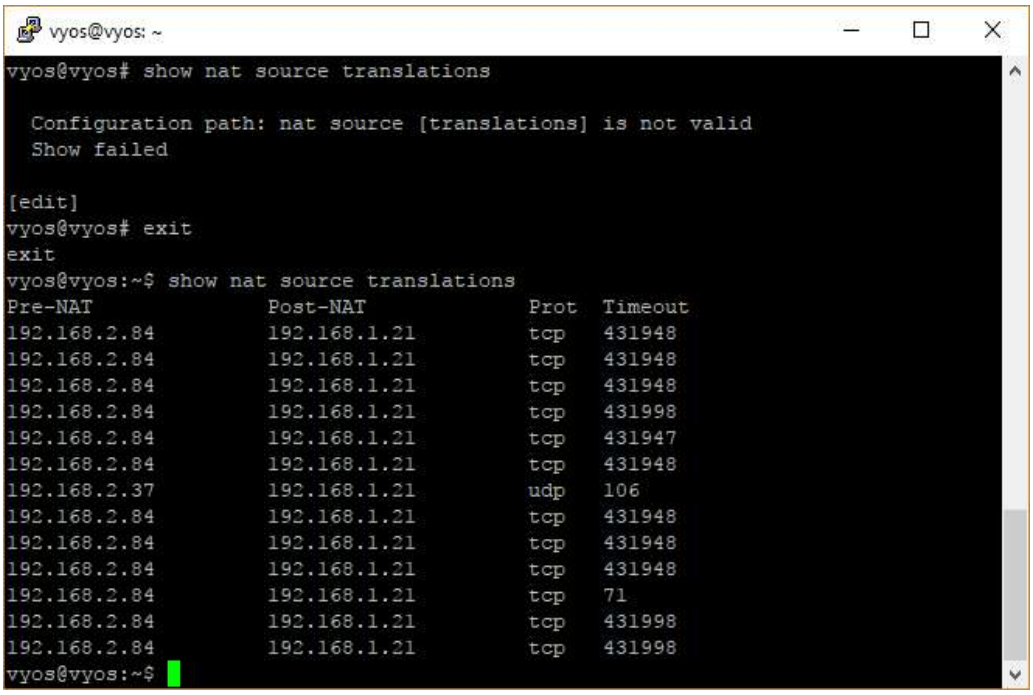
- 3 réseaux connectés via ce routeur:
  - 192.168.1.0/24 == >> le réseau dans lequel est ma machine physique. Dans ce réseau, c'est la livebox qui fait office de routeur pour tous les hôtes, y compris pour notre routeur VyOS.
  - 192.168.2.0/24 == >> le réseau no. 1 dans lequel seront des VMs virtual box que je vais créer. Dans ce réseau, c'est notre routeur VyOS qui fera office de routeur pour tous les hôtes (toutes les VMs que l'on va créer), avec l'adresse IP 192.168.2.1
  - 192.168.3.0/24 == >> le réseau no. 2 dans lequel seront des VMs virtual box que je vais créer. Dans ce réseau, c'est notre routeur VyOS qui fera office de routeur pour tous les hôtes (toutes les VMs que l'on va créer), avec l'adresse IP 192.168.3.1
  - Donc, si je PXE boot des Vms, il me faut configurer la config de isc-dhcp-server du pixie-node, pour que le serveur DHCP de la VM pixie bootée devienne celui de réseau no.2 ou no. 3... Vers un premier réseau unique, ce sera déjà un pas.
- ccc

- Pour faire que 2 Vms dans le réseau no. 1 (192.168.2.0/24), puissent avoir accès l'une à l'autre, il faut et il suffit (sans qu'elles aient accès à internet), qu'il y ait un routeur dans leur réseau. Ce routeur sera notre routeur VyOS, à l'adresse [192.168.2.23] . Pour que notre routeur VyOs agisse comme routeur dans ce réseau no. 2, et à l'adresse IP [192.168.2.23] , il nous faut configurer un des interfaces réseaux du routeur VyOS, pour qu'il ait une configuration IP avec {adresse= 192.168.2.23; netmask=255.255.255.0} :

```
set interfaces ethernet eth1 address 192.168.2.23/24
commit
save
# vérifiez, en exécutant la commande:
# ip addr|grep 168

# et voilà, le routeur VyOS agit comme routeur sur le réseau 192.168.2.23
# Il suffit de créer 2 Vms dans ce réseau 192.168.2.0/24, pour voir
# si elles ont accès l'une à l'autre (ping)
# Disons 2 Vms Ubuntu de config à ajouter dans [/etc/network/interfaces] :
# ------------------------------------------------------------------------
# VM1:
# auto enp0s3
# iface enp0s3 inet static
#       address  192.168.2.37
#       netmask  255.255.255.0
#       gateway  192.168.2.23
# ------------------------------------------------------------------------
# VM2:
# auto enp0s3
# iface enp0s3 inet static
#       address  192.168.2.84
#       netmask  255.255.255.0
#       gateway  192.168.2.23
# ------------------------------------------------------------------------
```

- En second temps, il faut faire en sorte qu'une VM dans le réseau no.1 ait accès au réseau internet. C'est bon, j'ai testé, et voilà le résultat du print de lhistorique des translations d'adresses sur le routeur:



  Ci-dessus, la trace de la traduction d'adressesayant eu lieu en ouvrant firefox et en accédant à google.com, depuis la machine `192.168.2.84`

  Jai deplsu testé que depusi ma amchine pphysique d'adresse IP 192,168,1,15, je ne peux évidemment pas pinger uen VM du réseau no.1, mais par contre je peux pinger ma machine physique à partir d'une VM du réseau 1 (j'ai testé à partir de `192.168.2.84 vers 192.168.1.15`).

- et la recette:

```
#############################################################
# 3./ On donne accès internet, aux machines sur le réseau 1:
#             192.168.2.0/24  [routeur: 192.168.2.23]
#############################################################
#
# Pour cela, on va définir une règle NAT
#
configure
###################################
# MAITENANT ON DEFINIT LA REGLE NAT
###################################
# 1./ on entre en mode édition d'une règle NAT
#     l'entier qui est le "numéro de règle (rule)" est libre de choix.
edit nat source rule 12
# 2./ on définit l'interface "OUTBOUND" ==>> donc pour
#     nous "DEHORS", c-a-d 192.168.1.0/24, soit [eth0]
set outbound-interface eth0
# 3./ on définit quelles adresses auront accès à l'interface
#     "OUTBOUND" (définit juste avant)
#     ici, je spécifie un réseau entier, mais il est possible
#     de spécifier d'autres manières, comme "tout sauf ce range
#     d'adresses".
#     ici, on donne donc accès à l'interface "OUTBOUND", à
#     toutes les VMs dans le réseau 192.168.2.0/24
set source address 192.168.2.0/24
# 4./ maintenant on définit la manière dont la
#   "translation" (traduction) d'adresses se fait
```

```
set translation address masquerade

commit
save
```

- Dernier test à effectuer: automatiser le provisionning eclipse, dans une VM, dans le réseau no. 1, et dans le réseau no. 2, une autre VM avec le provionning cible de déploiement, et vérifier que la VM eclipse a accès à la fois à la cible de déploiement, et à internet pour le repo github
- même chose que précédemment, sauf que je provisionne, en plus de la cible de déploiement, un gitlab privé, qui permet d'utiliser un repo git privé pour le repo assistant du deployeur-maven plugin.
- Dans les deux cas précédents, vérifier que l'on a pas accès à internet à partir d'une VM du réseau de l'infrastructure de déploiement, que 'lon a depusi le réseau de la VM eclipse, accès à la fois au réseau de la cible de déploiement, et à internet.
- ccc

J'ai déjà testé que la configuration minimale poiur faire communiquer 2 Vms en ip statique dans un réseau définit arbitrairement avec VyOS.Les 2 Vms n'ont pas accès à internet, et sont dans le réseau 192.168.2.0/24

La live box est dans le réseau 192.168.1.0/24

En l'état, 2 VMs et un routeur VyOS:

- VM ip statique: 192.168.2.37/24
- VM ip statique: 192.168.2.84/24
- routeur ip statique: 192.168.2.1/24

le test consiste à faire un ping d'une VM vers l'autre.
Ensuite je testerai si je peux créer 2 réseaux n'ayant ni l'un ni l'autre accès à internet, et ping d'un VM d'un réseau, vers une VM de l'autre réseau, grâce à une règle NAT que j'aurai configuré (les VM du réseau 1 ont accès aux VM du réseau 2, mais pas l'inverse, et puis les 2 sens)

ANNEXE II. Que la lumière soit

```
# ============== >>> le grand prinicpe <<< ==============

# ============== >>> le grand prinicpe <<< ==============

# ==>> donc 2 versionning:
#
#          ¤¤ [VERSIONNING DE LA RECETTE] un pour le code de la recette de construction de la cible de déploiement
#          ¤¤ [VERSIONNING DE LA DEPENDANCE] un repo git versionnant:
#
```

```
#                     ++ fichier docker-ompose.yml,
#
#                     ++ les fichiers dockerfiles et fichiers permettant de construire (avec un docker build par exemple) l'image
#                        customisée de chaque composant de l'infrastructure:
#
#                                 - [COMPOSANT SGBDR] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port):
#
#                                             le fichier dockerfile:
#
#                                                     FROM mariaDB
#                                                     ADD ./mon.fichier.de.conf.custom
#                                                     RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf
#                                                     CMD #là je crois c'est hérité du FROM
#
#                                             le fichier de conf custom mariadb: "./mon.fichier.de.conf.custom"
#
#
#
# - [COMPOSANT TOMCAT] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port)
#
#                      et en fait, le code de la recette de déploiement effectue un checkout d'une version bien donnée du fichier docker-compose.yml
#                      (et c'est là qu'est le lien entre le numéro de version de  la recette, et le numéro de version de la dépendance que constitue le fichier
[docker-compose.yml])
#
# ==>> donc, en réalité, 1 repo git de versionning de la recette , et N repo git de chaque dépendance (de degré 1) de la recette.
#
# -------------------------------------------
# le lien aux principes "the devops program":
# -------------------------------------------
#          ¤¤ une dépendance de degré zéro, cest: moi-même.
#          ¤¤ une dépendance de degré zéro de la recette, cest le code source de la recette elle-même.
#
#          ¤¤ une dépendance de degré zéro d'une appli java, c'est le code source de l'appli java elle-même.
#          ¤¤ une autre dépendance de degré zéro d'une appli java, c'est le code source/config du build de l'appli java.
#
# ------------------------------------------------------------------------------------------------------------------------------------------
#  !!!!!!!!!!!!!!!! et des roues imbriquées, petite roue dans grande roue, il y a une dimension supplémentaire avec les multiples repos GIT:
# ------------------------------------------------------------------------------------------------------------------------------------------
# Donc, dans l'arbre des dépendances, pour une dépendance de degré N:
#  - le versionning du code source de la dépendance de degré N, mentionne le lien aux versions des dépendances de degré N+1
#  - le versionning du code source de la dépendance de degré N, permet de changer le numéro de version d'une dépendance de degré N+1
# !!!!!!!! par exemple, pour changer le numéro de port de mariadb de la cible de déploiement
# !!!!!!!! (par config de mariadb et / ou changement de mapping docker numéro de port du conteneur docker ) :
#          # ## ON DETRUIT LA CIBLE DE DEPLOIEMENT
#          ¤ on fait docker-compose down
#          # ## ON EDITE DU CODE SOURCE
#          ¤ on fait une nouvelle version du fichier "docker-compose.yml" en changeant dans ce fichier le mapping du
#            numéro de port dans le   et/ou le numéro de version checkouté pour le checkout du repo versionnant le
#            dockerfile pour construire l'image {FROM mariaDB ADD ./mon.fichier.de.conf.custom RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf}
#          ¤ on édite sed le fichier "./docker-compose.yml", pour changer le mapping docker du numéro de port d'écoute de mariaDB avec un numéro de
#            port exposé par le conteneur du composant SGBDR
#          ¤ on édite sed le fichier de conf custom mariadb "./mon.fichier.de.conf.custom", pour changer le numéro de
#            port d'écoute de mariaDB ) "à l'intérieur du conteneur"
#          # ## ON COMMIT l'un ou l'autre, ou les deux...
#          ¤ [git add "./docker-compose.yml"]
#          ¤ [git commit -m "nouveau mapping docker numéro de port mariaDB -p noportexterne=noportinterne"]
#          ¤ [git add "./mon.fichier.de.conf.custom"]
#          ¤ [git commit -m "nouveau numéro de port mariaDB noportinterne="]
#          # ## ON TAGGUE
#          ¤ on détruit le tag ETAT_INITIAL_CIBLE_DEPLOIEMENT : [git tag -d vETAT_INITIAL_CIBLE_DEPLOIEMENT] (san détruire le commit associé au tag détruit)
#          ¤ on git taggue le commit  [git tag -a vETAT_INITIAL_CIBLE_DEPLOIEMENT -m "ETAT_INITIAL_CIBLE_DEPLOIEMENT: nouveau numéro de port etc..."]
#          ¤ on fait un git cdocker-compose up
#          ¤ on fait docker-compose up
#          # ## IL MANQUE LA PARTIE DOKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER IMAGES MONTEES ..?
#
#
le grand prinicpe: IL MANQUE LA PARTIE DOCKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER
IMAGES MONTEES
mais pour moi on pourrait ne pas changer les numeros de version
```

ANNEXE I. Configuration WIFI de la VM

- Pour ce faire, tu as besoin:
  - Du nom de la connexion Wifi: tu dois le connaître, car pour connecter ton PC/MAC en Wifi, tu as du rechercher le "nom de la connexion Wifi". Exemple de nom de connexion Wifi: "Livebox-3818". Plus précisément, c'est l'identifiant du point d'accès Wifi que tu utilises.
  - Et si la connexion Wifi n'est pas libre, d'autres informations, par exemple un mot de passe (c'est souvent le cas dans des lieux ouverts au public comme les bar et bibliothèques). Je vais supposer qu'un mot de passe est suffisant pour être autorisé à se connecter en Wifi, et si ce n'est pas le cas....: débrouilles-toi pour connecter ta VM en gros comme tu l'as fait pour connecter ton PC / MAC.

  Tu disposes donc de l'identifiant (le nom) du point d'accès Wifi, et du mot de passe. Pour configurer ton wifi:

  1. exécutes la commande:
     ```
     sudo gedit /etc/network/interfaces
     ```
  2. Regardes le contenu de ce fichier, tu devrais trouver un bloc qui ressemble à ceci (ici, j'ai mis "wlan0, mais ce pourrait être "wlan1", "wlan2", "wlan3" ...):
     ```
     auto wlan0
     iface wlan0 inet dhcp
     ```
  3. il y a peut-être deux lignes supplémentaires sous ce bloc (si elle n'y sont pas, ajoutes-les):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid etquelquechoselà
wpa-psk etautrechoseici
```

4. édites le fichier pour insérer l'identitifant (le nom) du point d'accès Wifi, et le mot de passe comme ci-dessous:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid l-identifiant-du-point-d-acces-wifi
wpa-psk et-le-mot-de-passe-ici
```

5. Nota Bene: les accès WIFI libres existent encore à certains endroits, et dans ce cas, n'ajoutes pas la dernière ligne (précisant le mot de passe):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid l-identifiant-du-point-d-acces-wifi
```

6. Enregistres (Ctrl +S), quitte gedit (Ctrl + Q), puis exécutes dans un terminal:

```
ip addr flush wlan0
systemctl restart networking.service
```

Ta VM devrait maintenant avoir accès à internet. Si ce n'est pas le cas, alors.. Eh bien tu vas te débrouiller, mais voici quelque liens qui pourraient t'aider:

https://doc.ubuntu-fr.org/wifi
https://doc.ubuntu-fr.org/wifi_ligne_de_commande

Bon courage! ;)

ANNEXE III. Configuration d'un data source pour applications Jee

- ## Pour Tomcat

  - Pour le cas datasource au niveau du serveur:

- [docker cp dans le conteneur] vérifer la présence de **`$CATALINA_HOME/lib/tomcat-dbcp.jar`** , le déployer si nécessaire
- [docker cp dans le conteneur] déployer dans **`$CATALINE_HOME/lib`** le jar du driver JDBC
- [docker cp dans le conteneur] configuration à appliquer dans le [**`$TOMCAT_HOME/conf/context.xml`**] (à vérifier par META-INF/context.xml du projet):

vierge, ce fichier a pour contenu:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
        license agreements. See the NOTICE file distributed with this work for additional
        information regarding copyright ownership. The ASF licenses this file to
        You under the Apache License, Version 2.0 (the "License"); you may not use
        this file except in compliance with the License. You may obtain a copy of
        the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required
        by applicable law or agreed to in writing, software distributed under the
        License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
        OF ANY KIND, either express or implied. See the License for the specific
        language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

        <!-- Default set of monitored resources. If one of these changes, the -->
        <!-- web application will be reloaded. -->
        <WatchedResource>WEB-INF/web.xml</WatchedResource>
        <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

        <!-- Uncomment this to disable session persistence across Tomcat restarts -->
```

```
        <!-- <Manager pathname="" /> -->

        <!-- Uncomment this to enable Comet connection tacking (provides events
                on session expiration as well as webapp lifecycle) -->
        <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
                /> -->
</Context>
```

et on y ajoute la balise <Resource>:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
        license agreements. See the NOTICE file distributed with this work for additional
        information regarding copyright ownership. The ASF licenses this file to
        You under the Apache License, Version 2.0 (the "License"); you may not use
        this file except in compliance with the License. You may obtain a copy of
        the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required
        by applicable law or agreed to in writing, software distributed under the
        License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
        OF ANY KIND, either express or implied. See the License for the specific
        language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

        <!-- Default set of monitored resources. If one of these changes, the -->
        <!-- web application will be reloaded. -->
        <WatchedResource>WEB-INF/web.xml</WatchedResource>
        <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

        <!-- Uncomment this to disable session persistence across Tomcat restarts -->
        <!-- <Manager pathname="" /> -->

        <!-- Uncomment this to enable Comet connection tacking (provides events
                on session expiration as well as webapp lifecycle) -->
        <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
                /> -->
        <!-- Configuration du datasource -->

<Resource name="organisaction/SourceDeDonnees" auth="Container" type="javax.sql.DataSource"
                                                                maxActive="20"
                                                                maxIdle="10"
                                                                maxWait="10000"
                                                                username="lauriane"
                                                                password="lauriane"
                                                                driverClassName="org.mariadb.jdbc.Driver"
                                                                url="jdbc:mariadb://localhost:8456/bdd_oraganisaction"
                                                                />

</Context>
```

et pour finir [docker restart du conteneur]

[Donc il me faut comme paramètres supplémentaires du plugin]

Ci-dessous, une correspondance entre les balises d'un pom.xml utilisant le plugin, et les variables d'environneemnt utilsiées dans les scripts de déploiement du datasource:

- MARIADB_JDBC_DRIVER_CLASS_NAME= <jdbc-driver-classname>org.mariadb.jdbc.Driver</jdbc-driver-classname>
- JEE_DATASOURCE_NAME= <jee-datasource-name>organisaction/SourceDeDonnees</jee-datasource-name>
- JEE_DATASOURCE_AUTH_USERNAME=<lx-user>lauriane</lx-user>
- JEE_DATASOURCE_AUTH_USERPWD=<lx-pwd>lauriane</lx-pwd>
- JEE_DATASOURCE_MAX_ACTIVE= <jee-datasource-max-active>20</jee-datasource-max-active>
- JEE_DATASOURCE_MAX_IDLE= <jee-datasource-max-idle>10</jee-datasource-max-idle>
- JEE_DATASOURCE_MAX_WAIT= <jee-datasource-max-wait>10000</jee-datasource-max-wait>

- JEE_DATASOURCE_URL_PREFIX=
  <!-- ici il y a le numéro de port et adresse IP SGBDR-->
  <jee-datasource-url-prefix>jdbc:mariadb</jee-datasource-url-prefix>

- JEE_DATASOURCE_URL=
  <!-- Non, inutile dans la configuration, l'URL peut
  être formée à partir:
  ¤ de l'adresse IP utilisée par le SGBDR
  ¤ du numéro de port utilisé par le SGBDR
  ¤ de la valeur de $JEE_DATASOURCE_URL_PREFIX <jdbc-datasource-url-prefix>
  ¤ de la valeur de $NOM_BDD_APPLI <jdbc-datasource-url-prefix>

  -->
  <jee-datasource-url>jdbc:mariadb://192.168.1.149:3306/db</jee-datasource-url>

- dans [WEB-INF/web.xml], ajouter une référence au datasource configuré dans le [$TOMCAT_HOME/conf/context.xml] ccc

```
  <resource-ref>
      <description>DB Connection</description>
      <res-ref-name>jdbc/TestDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
```

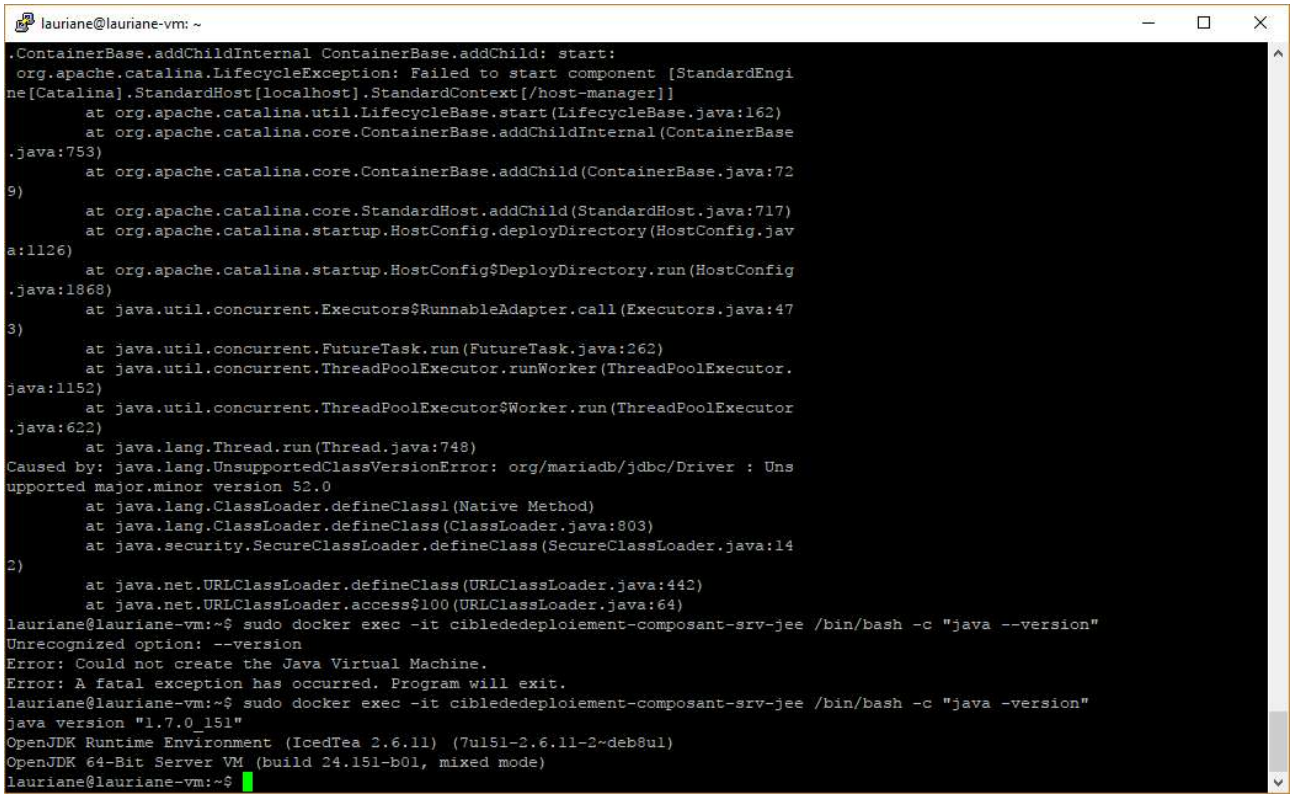
- Enfin, pour tester le datasource, il faut:
  - créer une table de tests "MembresAssos" dans la BDD, et y metttre quelques enregistrement. Exemple de table:

MembresAssos
prenom
nom
username
email
age

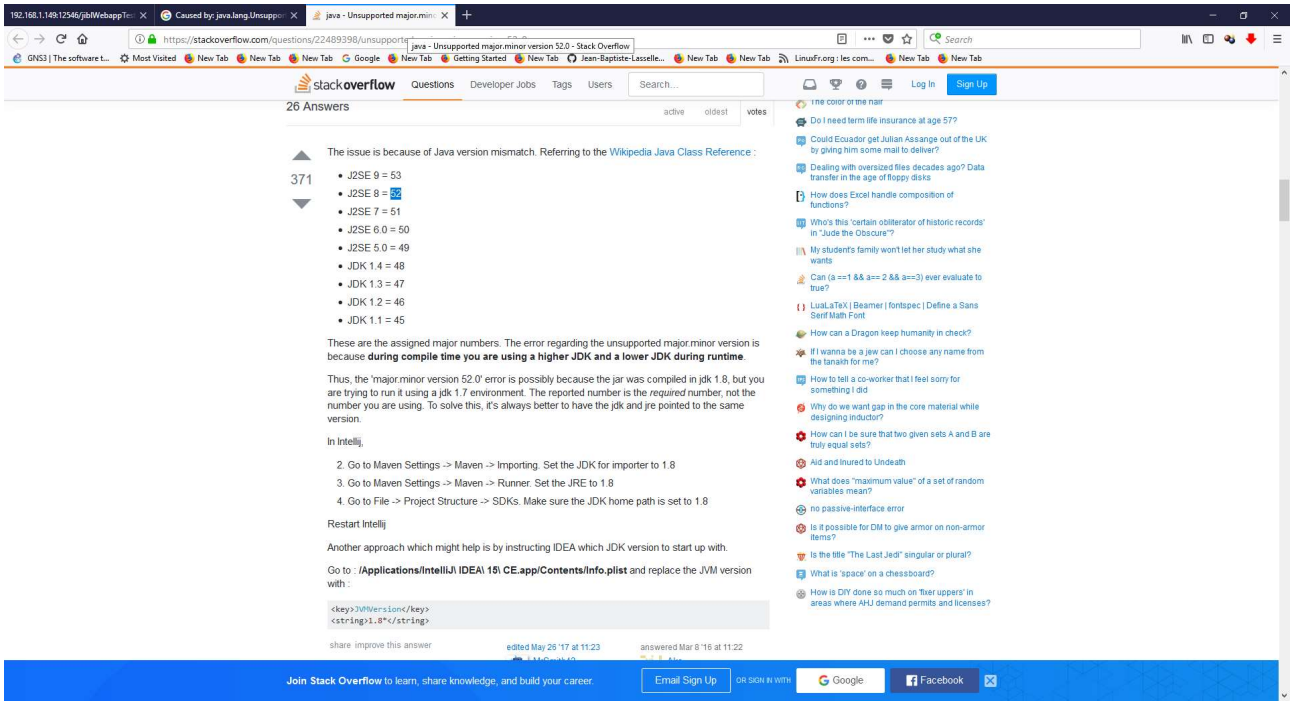
  -
    - Pour le cas datasource au niveau de l'application:

// à préciser, mais:

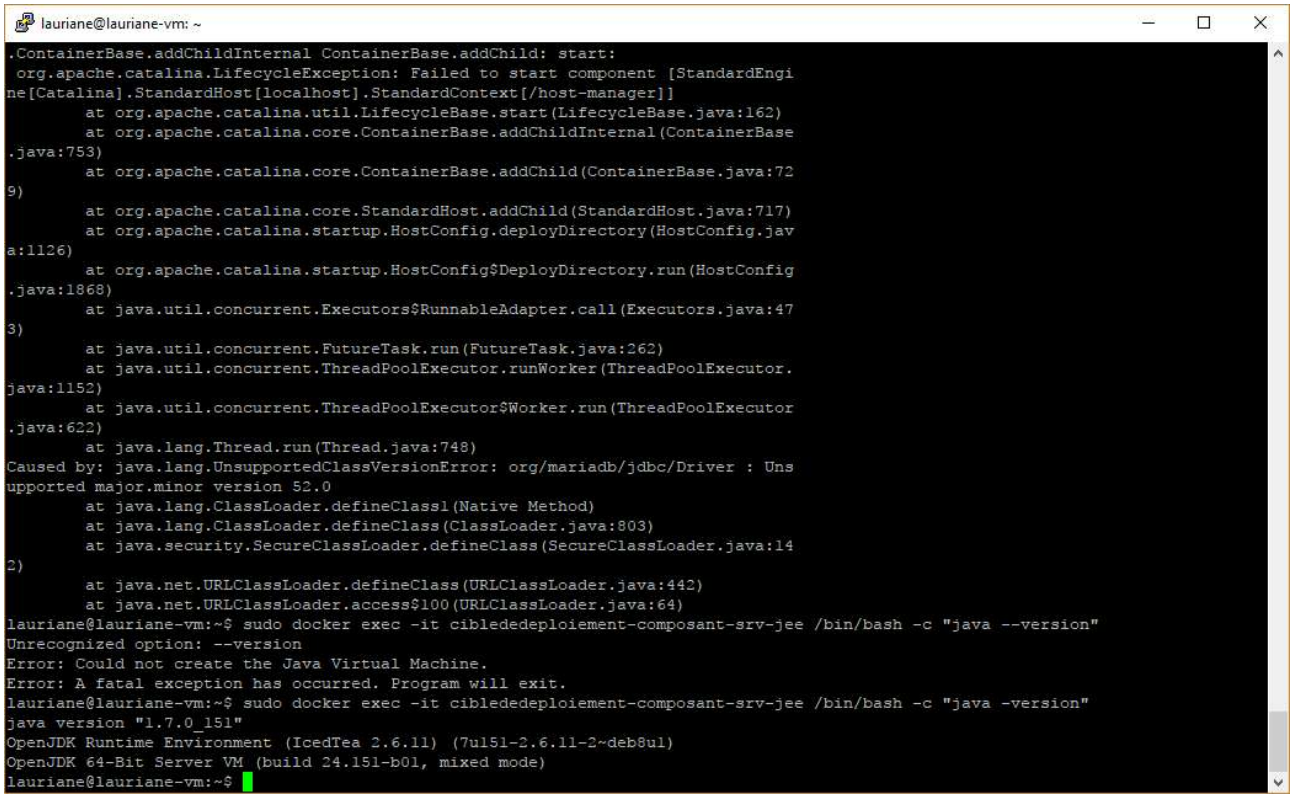
Donc j'ai quasiment finis d'automatiser le déploiment du datasource pour l'application
L'ereur sur laquelle je tombe, indiqe entre autre une erreur de compatibilité entre la version de JRE, et le driver MariaDB/JDBC:



Ci-dessus, ce sont les logs de tomcat dans le conteneur, et on voit une ligne :
"Caused by: java.lang.UnsupportedClassVersionError: org/maraidb/jdbc/Driver : Unsupported major.minor version 52.0"
Hors, le code 52 est associé au JDK 8 pour ce type d'exceptions natives Java, et elle est émise par la classe Pilote JDBC du pilote MariaDB/JDBC:

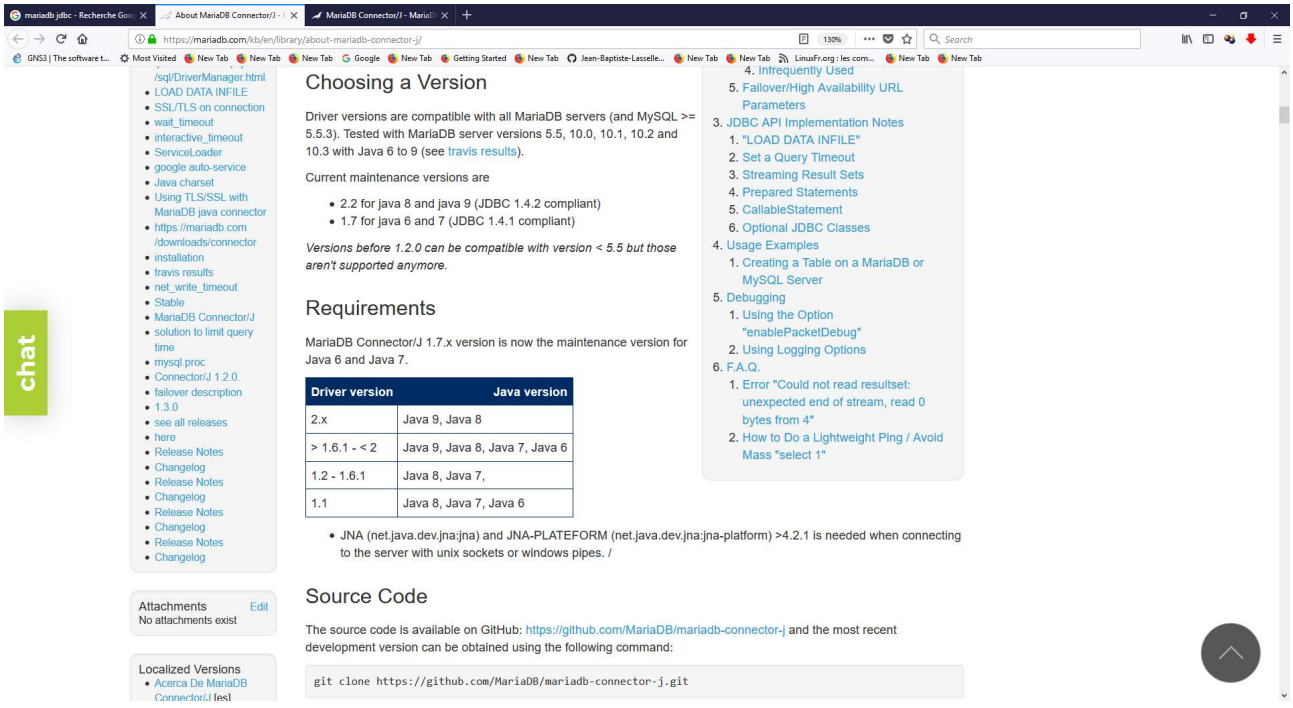


Bon, et de plus on voit que la versiond e java dans le conteneur qui edxécute tomcat est:



Donc il va falloir que je customise le dockerfile de la construction de tomcat. Dans ce dockerfile, je ferai l'installation de la JRE 8 au lieu de la JRE 7.

la dc officielle mariadb JDBC confirme l'incompatibilité (sachant que c'est la version 2.2.1 que je déployaisw pour le test) :



J'ai eut une autre déclinaison de mes problèmes d'incompatibilité avec la versiond e JRE dans le conteneur docker:

mes prores classes étaient compilées pour une cible JRE 8 (avec un source en Java 8), et ça re-levait une exception d'incompatibilité (et sans aucun log des exceptions de la webapp dans les logs serveurs):

J'ai testé ça en écrivant une simple Servlet faisant une réponse par une page HTML: en l'appelant directement avec son <url-pattern>, j'ai obtenu cette erreur d'incompatibilité entre la version de JRE, et la version de JDK que j'ai utilisé pour compiler mon code, avec les deux paramètres (source/target). Bref, j'ai changé la configuration du "maven-compiler-plugin", pour que mon code soit compilé en 1,7 en source, et en 1,7 en target, et le problème a été résolu, ma Servlet fonctionne (et permet de gérer les exceptions survenues dans l'application).

Arrivé là, j'ai modifié mon application pour qu'elle fasse un traitement des exceptions par config <error-page> dans le web.xml. Les exceptions sont attrapées par une Servlet, qui fait l'affichage d'autant d'infos que possible à propos de l'exception survenue. Une page jsp de test des exceptions permet de vérifier qu'une NullPointerException Levée, est bien traitée par ma servlet {@see `GestionnaireDexceptions` }. Tandis qu'aucune exception n'est attrapée lorsque j'invoque la page faisant usage de mon datasource configuré dans le web.xml

Oui, là, j'ai un problème de complexité pour mes tests:

version du driver jdbc /mariadb
version de tomcat
version de dbcp
version de Java dans le conteneur
version de Java en source et target de compilation (pom.xml)
fichier de configuration $CATALINA_BASE/conf/context.xml
fichier de configuration $CATALINA_BASE/conf/server.xml
fichier de configuration $MONPROJET/WEB-INF/context.xml

Si j'ai seulement 2 versions de chaque, j'ai un gros nombre de combinaisons à tester"

Je vais traiter ce problème plsu en profondeur, et en prenant en compte nos contraintes de projet en temps, à partir du tag: POINT_DE_RETOUR_COMPLEXITE_TESTS_TROP_LENTS

- **Pour Wildfly**